OVERFLOW DETECTION AND CLAMPING WITH PARALLEL OPERAND PROCESSING FOR FIXED-POINT MULTIPLIERS

BACKGROUND OF THE INVENTION

The present invention relates generally to the multiplication and clamping prediction of fixed-point multipliers. More particularly, the invention relates to a method and apparatus for increasing the speed of fixed-point data paths that involve multiplication of operands and parallel overflow detection and clamping based upon the magnitude of those operands.

Electrical circuits are routinely employed to perform arithmetic operations of operands represented by logical representations. Generally, it is desirable for arithmetic circuitry, and in particular multiplication circuitry, to have the fewest number of bits in order to perform the required calculations. Minimization of the required number of bits facilitates speed of the calculating circuit. However, counter-balancing the desire to have a limited number of bits to perform a given calculation is to avoid answer overflow. Overflow, or the situation in which an answer will exceed the number of bits designed for the answer, is not acceptable, as valuable valid data may be lost in performing the calculations. In circuits that perform calculations with a number of bits that may yield answers that overflow the set number of bits, clamping can be used to ensure that a result that overflows is clamped to a given acceptable value. Normally, the largest magnitude positive or negative representable number is employed as the overflow value. While the employment of a clamping operation is not always desirable, it is generally considered to be better than an overflow which may cause wrapping or undesired bits stored in the particular multiplication circuitry.

Typically, when clamping is desired, it is performed in a sequential fashion. In other words, the arithmetic operation is performed first, and when the result is available, it is then analyzed for overflow. If clamping is required, a clamping value replaces the computed value.

Referring now to Fig. 1, a prior system illustrates the serial processing in which the clamping analysis follows multiplication of particular operands. In this instance, operand 1

(reference numeral 2) and operand 2 (reference numeral 4) are input into an arithmetic operator 6. The output 8 from the arithmetic operator 6 is then directed into the overflow detection scheme 10. Any delays between arithmetic operator 6 and the overflow detection 10 are compounded, with each of their respective delays adding to the overall circuit delay. The clamp value 12 as determined by the overflow detection 10 is input along with the result 14 of the arithmetic operation 6, and a preset clamp value 16 into multiplexer 18. Multiplexer 18 selects the operation result 14 or the preset clamp value 16 in the instance of overflow detection based upon the logical level of clamp bit 12. The result is then output into register 20.

A major problem with sequential operation is that any delays, for both the arithmetic operation and the overflow analysis, tend to compound and therefore yield a relatively slow circuit to the extent that all delays are combined and added together to determine the total circuit delay. In any arithmetic operation, multiplication delays tend to be the largest. Therefore, there exists a need to implement a circuit and method in which arithmetic operations such as multiplication may be performed along with overflow detection. The result is a circuit that eliminates the compounded delay made up of the accumulation of each individual delay associated with sequential overflow analysis and operand processing.

SUMMARY OF THE INVENTION

The present invention provides overflow detection and clamping in parallel with multiplication of fixed-point multiplier operands that overcome the aforementioned problems, and provides a faster circuit than would otherwise be available from serial clamping analysis and arithmetic operation.

In accordance with one aspect of the invention, a method of detecting overflow in a clamping circuit includes inputting a first operand having a fixed-point format (i.e. represented by a fixed number of bits) into the clamping circuit and inputting a second operand having a second fixed-point format into the clamping circuit. A product overflow

output is determined based upon the first and second fixed-point format. The method includes predicting whether multiplication of the first operand with the second operand yields a result that exceeds the product overflow output, and performing at least partially the multiplication of the first and second operands. The determining step occurs substantially in parallel with the performing step.

In accordance with another aspect of the invention, a method of clamp detection is disclosed, and includes inputting a first and second operand to both a multiplier and an overflow detection circuit. The method includes multiplying the first and second operands to generate a result not to exceed a pre-determined number of bits, and determining an initial clamping predictor bit based upon the first operand and the second operand. The initial clamping predictor bit is logically ORed with a most significant bit of the result to produce a final clamping predictor bit.

Again, the multiplying and determining steps occur substantially in parallel.

In yet another aspect of the invention, a multiplication overflow detection circuit is disclosed. The circuit includes multiplication circuitry for at least partially multiplying a first and a second operand, overflow detection circuitry receiving the first and second operands that detects whether a result of the multiplication of the first and second operands exceed a maximum representable positive or negative value. The multiplication circuitry and the overflow detection circuitry operate substantially in parallel.

BRIEF DESCRIPTION OF THE DRAWINGS

The drawings illustrate the best mode presently contemplated for carrying out the invention.

In the drawings:

Fig. 1 is a schematic of a prior detection overflow scheme showing serial operand operation and overflow detection.

Fig. 2 is a schematic of parallel operation of the arithmetic operator and the overflow detection in accordance with the present invention.

Fig. 3 is a table illustrating necessary information in determining whether to clamp multiplication of the two operands in accordance with one aspect of the present invention.

Fig. 4 is a graph illustrating the simple overflow predictor and the regions necessary for additional overflow prediction calculations.

Fig. 5 shows a schematic of another aspect of the present invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

Referring now to Fig. 2, a clamping system 22 in accordance with the present invention is illustrated. Operand 1 is loaded into register 24 and operand 2 is similarly loaded into register 26. Registers 24, 26 are preferably flip-flops, but any suitable register capable of storing the operands are contemplated. Also, the operands may be passed directly from other arithmetic operators or other related logic, if suitable for the application. Operand 1 and operand 2 are loaded in a fixed-point format.

In fixed-point arithmetic, numbers are represented by zero or one sign bit, zero or more integer bits, and zero or more fractional bits. The integer and fractional bits can be a magnitude, a 1's complement, or a 2's complement value. The most common case is the 2's complement case. A short-hand description is used to denote how many bits there are in each category. This description is typically: <sign-bits>.<integer-bits>.<fractional-bits> or ***s.i.f.*** For a signed number with 4 integer bits and 5 fractional bits the notation would be 1.4.5. For an unsigned number with 6 integer bits and 3 fractional bits the notation would be 0.6.3, and often the leading zero is omitted to yield 6.3.

For a given fixed-point representation ***s.i.f*** there are associated numerical properties, (and assuming that the 2's complement is used):

Largest positive number representable: $2^i - 2^{-f}$

Largest (magnitude) negative number representable: -2^{i}
Smallest positive number representable: 2^{-f}
Smallest (magnitude) negative number representable: -2^{-f}

When fixed-point numbers are used in arithmetic operations such as additions, subtractions, and multiplications, the results generally require more bits to avoid overflow.

The rules for the two types of operations are:

1. Addition and subtraction:
   - If both inputs are of the same format ***s.i.f*** then the output requires ***s.(i+1).f*** to avoid overflow.
   - If the inputs are of different formats ***s.i1.f1*** and ***s.i2.f2***, then choose i = max(i1,i2) and f = max(f1,f2). Then the output requires ***s.(i+1).f*** bits to avoid overflow.

2. Multiplication:
   - If the inputs are of the same format ***s.i.f*** then the output requires ***s.(2*i+1).(2*f)*** to avoid overflow.
   - If the inputs are of different formats ***s.i1.f1*** and ***s.i2.f2***, then the output requires ***s.(i1+i2+1).(f1+f2)*** to avoid overflow.

Since multiplication most often produces the need for overflow detection and clamping, it is preferred that the present invention be utilized in multiplication of operand 1 and operand 2, with each operand in a fixed-point format. As Fig. 2 demonstrates, operand 1 and operand 2 are both supplied to the overflow detection 28 as well as the arithmetic operator or multiplier 30. The multiplier 30 takes operand 1 and operand 2 and determines at least a partial product of the binary operands. It is important to note that the multiplier does

not have to be full precision. It need only be twice the precision that is desired to be represented in the final result. For example, if it is desired to multiply two operands, each having eight bits, when the multiplication of eight bits and eight bits occurs, the answer will have sixteen bits of output. However, if it is only desired to have eight bits to come out in the final output, when the multiplication could actually generate up to sixteen bits, the multiplier will be used to generate nine bits. The nine bits represent twice the desired output range. Therefore, in the above-identified example, the answer will have a nine-bit result instead of the full sixteen-bit result. The final seven bits of the multiplication are not necessarily calculated.

Operand 1 and operand 2 are also presented to the overflow detection logic. The overflow detection 28 occurs in parallel with the multiplier 30 operation. It is the substantially simultaneous processing of the overflow detection and the multiplication that produces the efficiencies in processing time for the circuit. Also, it is the fact that the full product need not necessarily be calculated that results in further efficiencies in processing time.

The overflow detection circuit 28 considers the two input operands and predicts whether or not they will cause the product to overflow. The overflow prediction circuit 28 predicts when the product is going to be greater than or equal to two times the maximum desired value. This handles most of the clamping cases, but it is not exact. To get exact clamping, the product is computed to the (io+1)'th integer bit, and the most significant bit of the product 37 is used to finally determine whether or not to clamp.

The fixed-point format of the two inputs and the desired output must be known. These formats are denoted as:

Operand1: ***s.i1.f1***
Operand2: ***s.i2.f2***
Output: ***s.io.fo***

The product of the two input operands will require ***s.(i1+i2+1).(f1+f2)*** to avoid overflow. The assumption is that ***io*** < ***i1+i2+1***, (because, if ***io*** > ***i1+i2+1*** then an overflow can not occur).

Example

Consider the simple case of two positive input operands that yield a positive product. The maximum positive number that can be represented by the desired output representation is:

$$2^{io} - 2^{-fo}$$

Therefore, any combination of input operands that yields a product greater than or equal to the maximum value should be clamped to the maximum positive value:

$$\text{Output} = op1 * op2 \quad \text{if } (op1 * op2) < 2^{io} - 2^{-fo}$$
$$\text{Output} = 2^{io} - 2^{-fo} \quad \text{if } (op1 * op2) \geq 2^{io} - 2^{-fo}$$

The result 32 of multiplier 30 (which will in a preferred embodiment be a partial multiplication) is output from the multiplier 30. The detection overflow 28 outputs result 34 from its overflow detection circuitry, and as a result of a logical level of the clamp bit 34 it is determined whether clamping will occur. The most significant bit 37 (on line 36) of result 32 is logically ORed with clamp bit 34 through OR gate 38. Therefore, if any of clamp bit 34 or the most significant bit 37 is logically high, clamping will occur and a clamping signal 40 is output from OR gate 38. The clamping signal 40 is input into multiplexer 42. Result 32 of the multiplication is also input into multiplexer 42. Clamp value 44 is preferably hard-wired into multiplexer 42. The value of clamp value 44 is predetermined depending on the number of bits in operand 1 and operand 2. Preferably, the clamp value is the maximum representable positive or negative value. It is contemplated that in selecting a positive or negative value for

operand 1 or operand 2, the MSB (most significant bit) of each register 24, 26 could be exclusively ORed (XOR) together such that if the operands are of a different sign, the negative clamping value will be used for clamp value 44. Consequently, if both operands are of the same sign, the positive value for clamp value 44 will be used. Multiplexer 42 will select either result 32 or clamp value 44 depending upon the logical level of clamp signal 40, and will output the selection into output register 46.

Referring now to Fig. 3, a binary representation of several scenarios are given.

Take the case of two 1.6.4 operands, multiply them and return the product clamped to a 1.6.4 number. The maximum representable value in the output is $2^6 - 2^{-4} = 64\text{-}1/16 = 63.9375$. Some simple cases exist:

If op1 $\geq$ 32 and op2 $\geq$ 2 then clamp (45a)
If op1 $\geq$16 and op2 $\geq$ 4 then clamp (45b)
If op1 $\geq$ 8 and op2 $\geq$ 8 then clamp (45c)
If op1 $\geq$ 4 and op2 $\geq$ 16 then clamp (45d)
If op1 $\geq$ 2 and op2 $\geq$ 32 then clamp (45e)

The binary representation of these scenarios is given in Fig. 3.

It can be seen that the number of leading zeros in the integer portion of the operands is indicative of the magnitude of the operands, and by adding the number of leading zeros of both op1 (47) and op2 (49), there is a constant number of leading zeros 51. Therefore, the fixed-point format of the operands will determine the constant number of leading zeros to determine whether clamping occurs. Clamping must occur when:

If (<op1 leading zeros> + <op2 leading zeros>) $\leq$ 4 then clamp
Else don't clamp

This relatively simple predictor works for a substantial portion of products, but it is not completely accurate. If only this predictor were used, it would be possible to get products that could be as much as twice the desired max value, i.e.,

(max product given simple predictor) <2 * (desired max value)

Therefore, to get an accurate clamp predictor, the above simple predictor is used, and the multiplication generates a result that is of format s.(io+1).fo. Then, for positive operands, an overflow is detected by ORing together the MSB of the integer bits and the clamp-prediction.

Referring now to Fig. 4, a graph is shown showing the regions of products that are clamped by the initial clamp predictor. The broad region covered by the simple, initial clamp predictor is shown generally by the numeral 50. Regions 52 represent those products that require the io+1 integer bits of the product to get accurate overflow detection and therefore require more precise clamp prediction. Region 51 represents the region where clamping is not needed because the product does not exceed the desired number of bits. However, this graph is representative of only one particular set of operands (both positive). Other predictors will produce different data depending upon the signs of the operands.

General Case

The general case consists of three subcases depending on the signs of the input operands:

Case 1: (op1 > 0 and op2 > 0)
Case 2: (op1 < 1 and op2 < 1)
Case 3: (op1 > 0 and op2 < 1) or (op1 < 0 and op2 > 0)

Each of these cases will be examined in turn.

Case 1: Both operands are positive

The simple clamp predictor is:

If the sum of the input operands' ***leading zeros*** is less than or equal to

$$(i1 - io) + (i2 - io) + (io - 2)$$
$$= i1 + i2 - io - 2$$

then the circuit must clamp.

The accurate clamp prediction must use the (io + 1) bit of the product, i.e., the product must be computed at least to (io + 1) integer bits. This bit is ORed logically with the simple clamp predictor to yield the accurate clamp determination.

Case 2: Both operands are negative

When both operands are negative, their product is positive. Therefore, if an overflow case exists we clamp to the same value as mentioned before, namely:

$$2^{io} - 2^{-fo}$$

When both operands are negative the simple clamp predictor must count ***leading ones*** in the input operands. If the sum of the input operands' leading ones is less than or equal to

$$(i1 - io) + (i2 - io) + (io - 1)$$
$$= i1 + i2 - io - 1$$

then the circuit must clamp.

However, there is an additional case, when both operands have only zeros after the leading ones (e.g., 111100.0000) then the simple clamp predictor should also clamp. This is like counting leading zeros on a bit-reversed version of each input. If the trailing zeros plus

the leading ones equal (i+f) for both inputs then clamp. As before, the accurate clamp prediction must use the (io + 1) bit of the product, i.e., the product must be computed at least to (io + 1) integer bits. This bit is ORed logically with the simple clamp predictor to yield the accurate clamp determination.

Case 3: Only one of the operands is negative

When only one of the input operands is negative, their product will be negative. Therefore, if an overflow case exists we clamp to the largest (magnitude) negative value, namely, -2^{io}.

In this case, the simple clamp predictor must count ***leading ones*** for the negative input, and ***leading zeros*** for the positive input. Then, if the sum of the inputs' leading ones and leading zeros is less than or equal to

$$\begin{aligned} & (i1 - io) + (i2 - io) + (io - 2) \\ = \; & i1 + i2 - io - 2 \end{aligned}$$

then the circuit must clamp.

As before, the accurate clamp prediction must use the (io + 1) bit of the product, i.e., the product must be computed at least to (io + 1) integer bits. This bit is logically inverted, then ORed logically with the simple clamp predictor to yield the accurate clamp determination.

Referring now to Fig. 5, another embodiment of the present invention is shown. In this embodiment, the clamp bit may be stored in a clamp bit register 60 and the result of the multiplier operation may be stored in multiplier result register 62. In this embodiment, after

one of the result 62 and the clamp value 64 is selected, other logic 68 is introduced to process the output 66 as part of a pipelining stage. The pipelining stage enables further processing of a partial result if the computation is not completed in a single clock cycle. The clamp prediction along with a partial result is then saved for the following clock cycle where completion of the clamping prediction will occur in the second clock cycle.

The initial clamp predictor may be utilized as soon as the two operands are available. The final clamp predictor must occur after the result of the multiplier has completed its partial computation to the appropriate necessary bit. Therefore, the simple clamp prediction occurs at the same time as the multiplier as manipulating operand 1 and operand 2.

Although two operands are shown, it is contemplated by the present invention that any number of operands may be used as inputs to the multiplier operation. In addition, the multiplication may occur in several stages with other multiplications coming before or after in a similar manner. The cloud of logic represents other multiplications, other additions or other logic operations on the result 66 of the multiplexer.

The present invention has been described in terms of the preferred embodiment, and it is recognized that equivalents, alternatives, and modifications, aside from those expressly stated, are possible and within the scope of the appending claims.